(12)

INTERACTIVE AIDS FOR CARTOGRAPHY AND PHOTO INTERPRETATION

Semiannual Technical Report

Covering the Period 12 May 1977 through 11 November 1977

December 1977

By: Harry G. Barrow, Principal Investigator
(415) 326-6200, Extension 5089

Sponsored by:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Contract DAAG29-76-C-0057

ARPA Order No. 2894-5

Program Element 61101E

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200
Cable: SRI INTL MPK
TWX: 910-373-1246

SRI International

SRI International

6 # INTERACTIVE AIDS FOR CARTOGRAPHY AND PHOTO INTERPRETATION.

9 Semiannual Technical ~~Report~~. 12 May-11 Nov 77,

Covering the Period 12 May 1977 through 11 November 1977

11 Dec~~ember~~ 1977

12 53 p.

10

By: Harry G. Barrow, Principal Investigator
(415) 326-6200, Extension 5089

Sponsored by:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

15 ~~Contract No~~: DAAG29-76-C-0057,
✓✓ ARPA Order [redacted] 2894 [redacted]
Program Element 61101E

Effective Date: 12 May 1976
Expiration Date: 9 October 1978
Amount of Contract: $521,363

SRI Proj~~ect~~ 5300

Approved:

Peter E. Hart, Director
Artificial Intelligence Center

Earle D. Jones, Executive Director
Information Science and Engineering Division

SRI International

410 281 JOB

ABSTRACT

The ARPA Image Understanding Project at SRI has the scientific goal of investigating and developing ways in which diverse sources of knowledge may be used to interpret images automatically. The research is focused on the specific problems entailed in interpreting aerial photographs for cartographic and intelligence purposes. A key concept is the use of a generalized digital map to guide the process of image interpretation.

In the past six months, the components developed under this project have been integrated into a single system, known as Hawkeye. The system includes modules for handling a large knowledge base in the form of a semantic net and terrain data files for using a raster-scan display and graphics tablets, for running task-specific subsystems, and for communicating with the user via graphics and in English. With the Hawkeye system, a user (or a program) can retrieve and display digital images, calibrate them in terms of world coordinates, make measurements from them and ask questions and give commands in English such as: What is this road?, Show me the Transamerica building, or How many photos are there of Alameda? These capabilities are fundamental, both for interactive and fully automated image analysis.

CONTENTS

ILLUSTRATIONS

TABLES

ACKNOWLEDGMENTS

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office under Contract No. DAAG29-76-C-0057.

The work reported here was carried out by H.G. Barrow. R.C. Bolles. T.D. Garvey. J.H. Kremers, K. Lantz.[*] J.M. Tenenbaum. and H.C. Wolf.

----

[*] Visiting from the Department of Computer Science. University of Rochester.

I INTRODUCTION

A. Overview of the Project

This report describes the ongoing ARPA Image Understanding Project at SRI. The central goal of this project is to investigate and develop ways in which diverse sources of knowledge may be brought to bear on the problem of interpreting images. The research is focused on the specific problems of interpreting aerial photographs for cartographic or intelligence purposes. Up to a certain point, cartography and photo interpretation overlap considerably: In making a map from aerial imagery, the images must first be interpreted; in interpreting imagery, a map of the area is a very valuable tool. Both classes of activity demand a basic level of image handling and map knowledge, much of which can be successfully automated.

A key concept in our work is the use of a *generalized digital map* to guide the process of image interpretation. This map is actually a data base containing generic descriptions of objects and situations, available imagery, and techniques, in addition to topographical and cultural information found in conventional maps. The data base can be used simply as a source of specific information about an area, as a source of general information about relationships and characteristics of classes of object, or in conjunction with imagery to answer questions about dynamically varying facts.

We recognize that it is not possible to replace a skilled photo interpreter within the limitations of the current state of image understanding. It is possible, however, to facilitate his or her work greatly by providing a number of collaborative aids that alleviate the more mundane and tedious chores [1].* Our aims, therefore, have been:

----
* References are listed at the end of the report.

* To identify specific ways in which computers can be of assistance
* To develop interactive or automatic programs that demonstrate feasibility
* To integrate the programs into a coherent framework that can evolve toward the goal of automated photo interpretation as more advanced capabilities are developed and incorporated.

Our earlier work, described in previous progress reports [1] [2] [3] concentrated on the first two goals. We have demonstrated useful programs for a number of common tedious or arduous tasks, including monitoring, measuring, counting, and tracing. The demonstration programs were independent, however, sharing only data files. We are now integrating basic and more advanced capabilities into a single coherent system, known as Hawkeye.

B. Overview of the Hawkeye System

Hawkeye is intended to provide a complete environment for a wide variety of photo interpretive or cartographic situations. Like PACER [1], it includes image-handling facilities, map display facilities, and an on-line data base of intelligence information. In Hawkeye, however, these facilities are all on line and tightly integrated. Many basic photo interpretation functions can be perfomed interactively, on images and maps in both printed and digital form. In addition, some selected higher-level functions can be performed automatically by modules that exploit knowledge in the data base to guide image interpretation.

The working area, shown in Figure 1, contains a video terminal for communicating in conventional fashion with the computer, a color display monitor with a trackball for presenting digitized pictures and maps, and a digitizing table and cursor for using pictures and maps in document form. Users communicate with Hawkeye naturally, in free-form English and via interactive graphics.

The system organization is modular, the primary components being display console and digitizing table utilities, a generalized map data

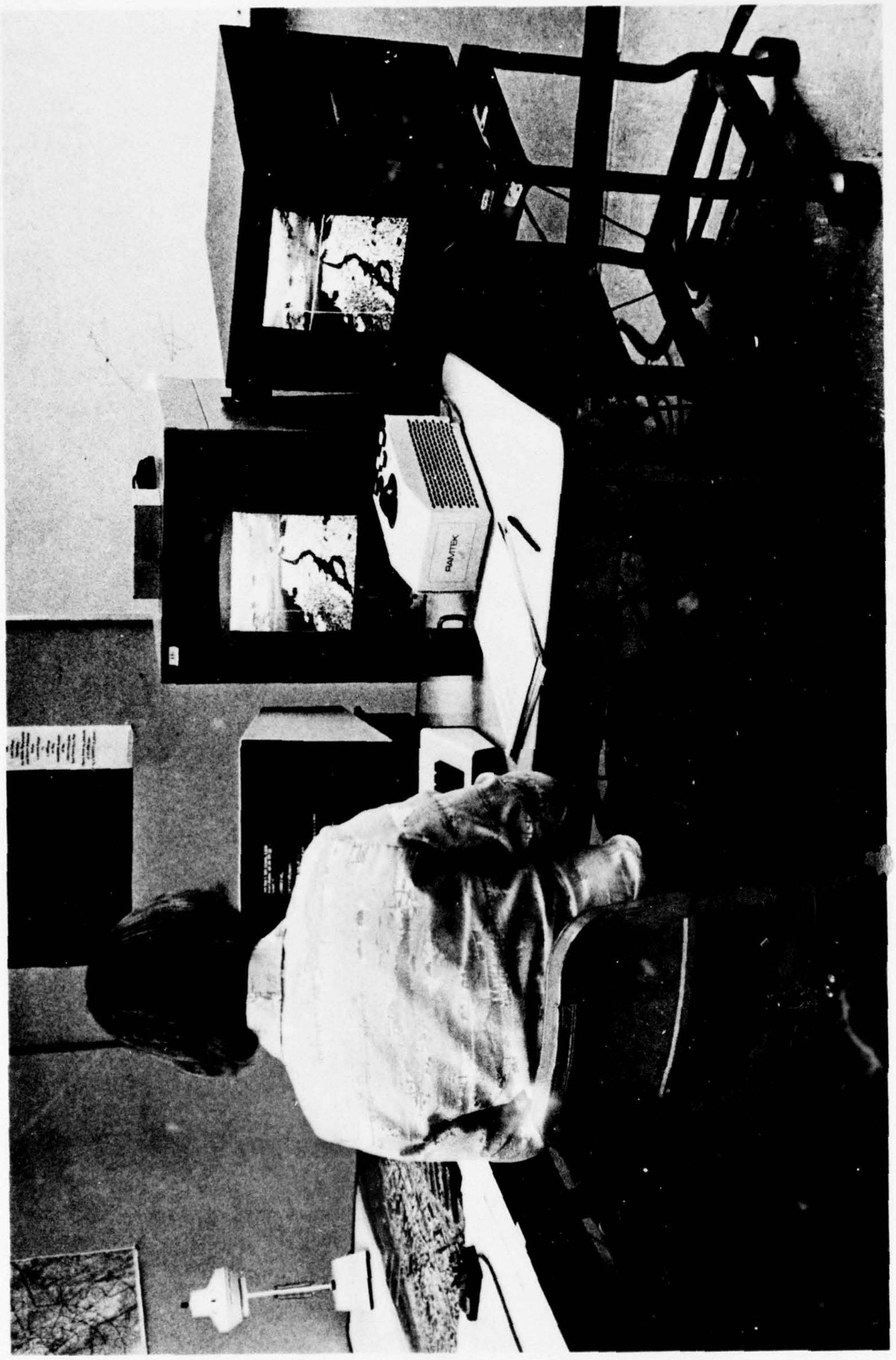

FIGURE 1 HAWKEYE WORK STATION

base, an image library, general image-analysis routines, task specialist routines, and the necessary user interface. Table 1 outlines the function of the various modules currently in use.

The following scenario illustrates the capabilities that will be available when integration of the facilities we have already demonstrated is complete.

A user entering Hawkeye encounters a task queue containing requests (from hypothetical analysts). Retrieving a task, the user can query the data base to determine the best available photo coverage. That image can be retreived and displayed and, if the image has been calibrated, selected features from a digital map can be overlayed upon it. Relevant reference documents in hard copy form (e.g., photos and maps) can be mounted on the digitizing table and put into correspondence with the digital data.

Correspondence with the map data base is established interactively by designating control points in the uncalibrated image or map. Calibration of digitized images could be performed automatically, with the system selecting potentially visible landmarks (using navigational data associated with the image) and then locating them in the image using scene analysis techniques.

An important class of task is confirming the validity of existing knowledge. The completed system will be able automatically to verify the presence of certain cartographic features, such as roads and waterways, and also to monitor the status of some typical dynamic situations, such as ships berthed in harbor or boxcars stored in a classification yard.

Another class of task is concerned with identifying new features and adding them to the data base. This can be accomplished using interactive aids for mensuration and tracing. Cartographic features from printed documents must be digitized manually by tracing them on the tablet. Elevations can be obtained automatically if they are available in the terrain data base. Numerical annotations, such as the heights of

Table 1

HAWKEYE SYSTEM MODULES

Module	Functions
User interface	• Main means of communication with the user • Communication via natural language • Command interpreter • Help facility
Display	• Shared access to display by multiple modules • Presentation of pictorial information (pictures and maps) • Graphical communication (pointing, indicating)
Graphics tablet	• Digitization of pictorial data (maps, photos) • Graphical communication (pointing, menus)
Generalized map knowledge base	• Repository of cartographic and cultural data • Generic definitions of semantic objects • Image library and index • Question answering about data base
Terrain data base	• Topographic data • Question answering about topography
Map/image correspondence	• Determination of camera and digitization parameters • Determination of transformations between map and image • Mensuration (length, height, distance)
Task specialists	• Road Tracing • Railyard monitoring • Harbor monitoring

structures, are entered via a keyboard. When the source of information is a digitized image, the user can invoke the interactive tracing aids that require from the user only a crude guideline. Dimensional measurements can be obtained interactively using various mensuration aids.

The user can obtain information from the data base to aid him in his interpretations. The system can answer simple queries, such as: Show me Pier 14, What is this building?, or How high is that mountain?. These queries are entered in English via a keyboard and via a display cursor. In this way, the user can determine the identity of known structures, or whether a given structure is new, obtain intelligence background information, indicate features to the system, or have the system indicate features (such as all the roads in an area). The analyst, similarly, could query the data base, which thus serves as a channel of communication between the analyst and the interpreter.

The system can be programmed to answer more complex queries, such as: How many ships were in Oakland harbor yesterday?, by retrieving the relevant image from the library and invoking an appropriate task specialist. The system has the ability to accept such requests entered remotely (say, by intelligence analysts) and execute them automatically if it understands, or else relay them to the user (the photo interpreter) for interactive execution.

The questions that can now be handled automatically are limited by the present small size of the data base and the available specialist routines, which automate tasks carefully chosen to exploit existing primitive low-level vision capabilities. However, demonstrated capabilities do show the potential of bringing image understanding and artificial intelligence approaches to bear on problems in cartography and photo interpretation.

II THE HAWKEYE SYSTEM

A. System Organization

The Hawkeye system is necessarily a large complex of programs that make considerable demands upon the supporting hardware and software. It contains sufficient program and data to fill the available user address space of 256K words on a DEC PDP-10 several times over. Very efficient code is required for low-level mass processing of image arrays, but flexible symbol manipulation is required for the higher levels of processing. There must be efficient communication between the various levels and subsystems. The system is highly interactive, using several very different devices, with parts that must be sharable among several users, while other parts must be user-specific.

Unfortunately, the requirements are not met by currently available combinations of hardware and software. In particular, no single existing programming language is truly adequate for advanced image understanding work; hence it is necessary to program in a mixture of languages. Because of this, and because of its total size, the system must be partitioned into subprograms, or modules. We have attempted to modularize the Hawkeye system in a way that retains as much flexibility as possible, while sacrificing as little power as possible. The modules correspond to parts of the system that perform a distinct set of operations, are programmed in a single language, and communicate with the user mainly via one device. Some modules (e.g., the data base) could also be shared by several users.

The Hawkeye system currently consists of the modules outlined in Table 1. The various modules and their capabilities are discussed in more detail later. In this section, we address the problem of overall organization of the system.

The modules of the Hawkeye system are implemented as independent processes (forks), each written in an appropriate language (INTERLISP, SAIL, FORTRAN, or MACRO) with its own data structures. Processes interact by sending and receiving interprocess messages, and are organized as in (Figure 2). In an early version of the system, interaction was purely vertical, between the user interface and a particular subservient module. We found, however, that such a scheme was not flexible enough.

Each process performs a specific set of functions, either at the request of other Hawkeye processes (e.g., the display handler) or in the direct interests of the human user (e.g., a top-level task specialist, such as the railyard monitor). The former processes are "server processes," whereas the latter may be classified as "user processes" (although occasionally some processes might behave as both user and server). A server process is associated with each external connection (i.e., device or data base). Each server presents a standard interface to the rest of the system. Thus, knowledge of the idiosyncrasies of a particular device or data base is required only within the process dedicated to it.

The user communicates with the system primarily via the user interface module (written in INTERLISP), which then calls upon appropriate server modules to carry out his request. The user interface module is also responsible for initializing the server modules when the system is first started.

This organization of the system is very convenient. It permits easy extension of the system by adding modules, and even the incorporation of independent programs within the Hawkeye framework. It also permits using less than the full system (e.g., just the user interface and data base) or sharing of modules among users, if desired. Such flexibility is valuable both for building and using complex systems.

The following sections describe intermodule communication and the supporting modules in more detail.

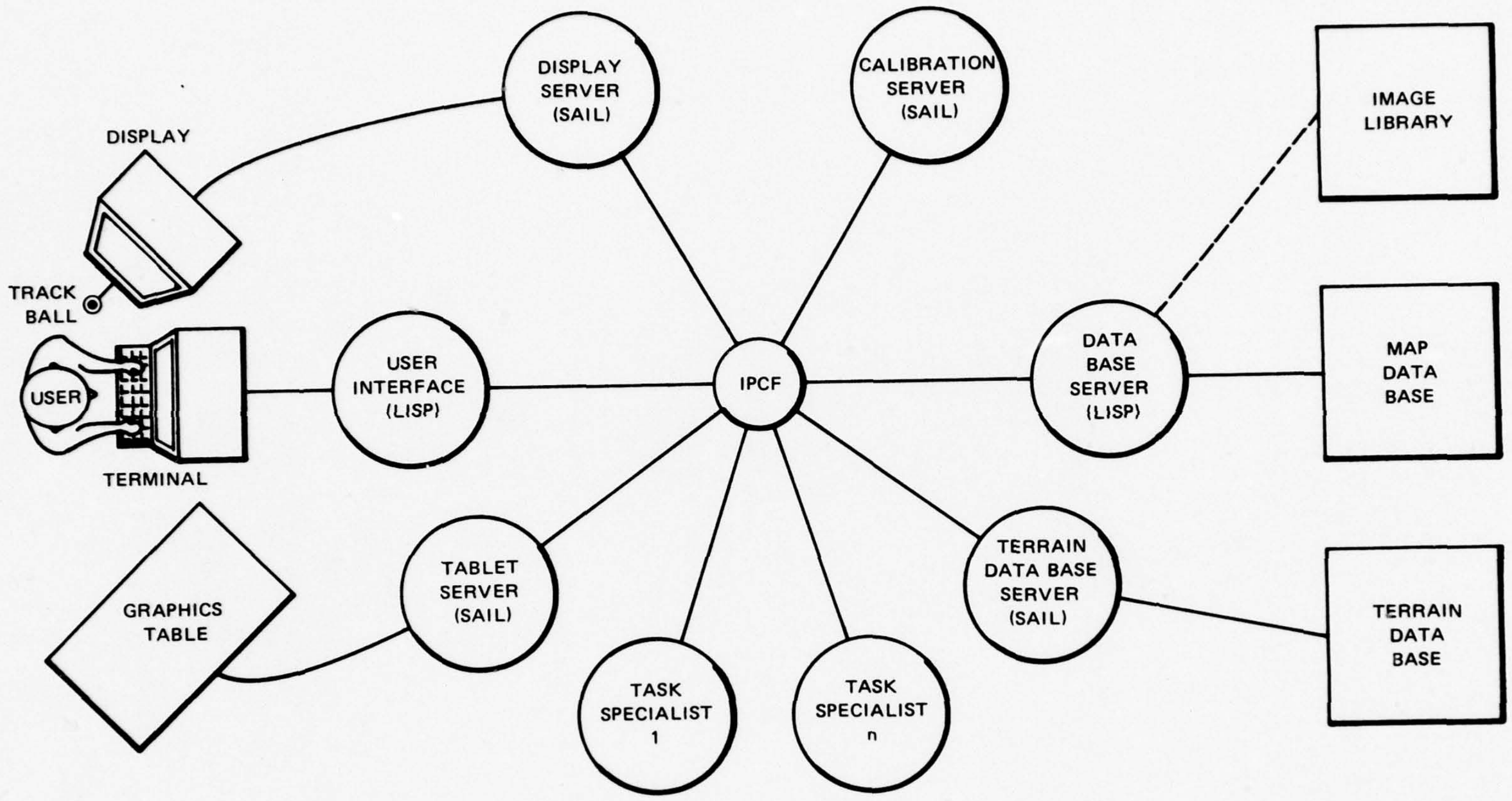

FIGURE 2 SYSTEM ORGANIZATION

A conventional self-contained program of modest size uses two principal routes for passing information among its components: via arguments to subroutines, and via global variables. However, a large system that consists of many modules that are essentially independent programs cannot use these methods for intermodule communication, since modules occupy different address spaces. In this case, modules must communicate either via shared files or whatever other means are available in the operating system for this purpose.

The TENEX and TOPS-20 operating systems provide several methods of communication between programs: files, pseudoteletypes, shared pages, and the Inter-Process Communication Facility (TOPS-20 only). Files have the disadvantage of being primarily sequential, and of entailing significant overheads for opening and closing. Pseudoteletypes are strictly sequential, character-passing channels. They enable the two modules concerned to treat communication like terminal I/O, but require parsing routines at each end to translate from text format to internal format.

Our first version of the Hawkeye system was implemented under TENEX and made use of the page-sharing facility. Page sharing is really sharing a block of memory, or sharing global variables. An additional facility is needed to permit control to be passed between modules: In this case, the fork (subjob) machinery was used. The system organization was hierarchical, with a top-level module and a set of inferior modules (forks). Inferior modules were treated much like subroutines, with arguments being passed via shared pages and control being handed temporarily to the inferior module, which eventually returned it to the top level module.

When the system became more complex, the hierarchical organization became insufficient: It was necessary for control and information to be passed among the inferior modules. We contemplated extending the shared-page scheme, but there were a number of problems: Allocating one shared page for each pair of modules is expensive and makes system extension difficult, while allocating one (or more) pages common to all modules leads to interlock and scheduling difficulties. Sharing pages does not readily permit communication between concurrent processes -- that requires some form of queueing of messages.

In the current version of Hawkeye, interprocess communication is implemented using the Inter-Process Communication Facility (IPCF) of the TOPS-20 operating system. This facility provides significantly better interaction than the pseudo-teletype and shared-page facilities to which we were limited under TENEX. IPCF enables processes (including forks and jobs) to send and receive messages in the form of packets, up to a page (512 words) in length. Messages are copied from the sender's address space and placed in an input queue in system space. The packet remains in the queue until the receiver requests it, at which time it is copied to the receiver's address space.

Messages consist primarily of requests and responses with the format:

Source ID : Destination ID : Message ID : Message Data

The source and destination IDs are the unique process identifiers assigned to the associated processes at the time they were created. Each module may contain several processes that are independent or that share memory, subroutines, or other facilities. In Hawkeye, the message ID conveys the type of processing and information required of the receiver, and thus implicitly how the message should be unpacked (i.e., as integers, reals, characters, or some mixture).

Once a request is posted, the requester (sender) usually waits until the requestee (receiver) has finished processing the request. The requestee, on the other hand, is typically a server and operates by waiting for a request, processing it to completion, sending a response, waiting for the next request, and so on. Servers are not interrupted by incoming requests while processing a previous request. However, they may themselves send requests to other servers and await replies before proceeding. For this purpose, each process has two channels of communication, each with its own ID: One channel is for receiving requests and sending responses, and the other is for sending its own requests and receiving answers. A process may choose whether it will respond to requests while awaiting an answer, but it had better do so if a recursive situation can arise. Processes are used in Hawkeye not so much to achieve parallelism, but to acquire a larger address space and to integrate multiple languages.

Interface routines are needed to match the basic message-passing machinery to the various programming languages used in implementing Hawkeye modules. Each module must know what computations can be performed by other modules, and the data format of the message. The send interfacing routines construct a message, packing the required information, possibly after type-conversion. The receive interfacing routines unpack the information and call an appropriate subroutine. For example, LISP interfacing routines will accept arbitrary objects (e.g., lists, arrays, numbers) for transmission, convert them to the form expected by the receiver, pack, and transmit them. One such routine accepts any LISP S-expression and transmits it to any other LISP process, which then evaluates it and returns the result. The user interface module, for example, can construct S-expressions from English input and transmit them to the data base module for evaluation.

It appears that the Inter-Process Communication Facility, with our extensions to it, is powerful, flexible, and convenient. It has the additional advantage that it permits communication between processes in different time-sharing jobs. Thus, it is possible for users to share facilities, (such as the data base) and to communicate among themselves. For example, it would be possible for two photo interpreters using the IPCF machinery to exchange example images and even to converse about them.

B. Display Server

The display server is the channel through which all graphical information communicated via the display must pass. Not only does this restriction avoid duplication of display-handling code in other modules. but it centralizes knowledge of the state of the display screen. The display server can allocate subsets of the display capabilities to requesting processes and keep track of what is visible to the user: Bookkeeping of this sort is essential. for example. when the user is attempting to point at a feature displayed on the screen.

The basic capabilities of the display hardware are as follows: The screen area is 256x256 pixels, and grey-scale images can be displayed with four bits of brightness resolution in red. green. or blue. Images may thus be displayed in color or monochrome. There are also two one-bit overlays in red and green. Vectors may be drawn in any combination of colors, including the overlays. A trackball is also available for locating a point on the display screen or tracing a sequence of points.

The user. or a process. can define display windows. A display window is a rectangular portion of the display screen. usually indicated by pointing with the track ball cursor, and some subset of the available colors (Figure 3). Grey scale images or line drawings may be subsequently displayed within the confines of the window. without affecting the rest of the visible display. The user may. for example, use one window as an inset for displaying an enlarged portion of an image (Figure 4). or two windows for displaying two different images. and may overlay the map on any of them. When several windows overlap, the most recently displayed image overwrites the others. and the display server records this fact by keeping a list of the windows in use. ordered by recency of display (Figure 5). When the user points with the trackball cursor, the server searches this list until it finds the most recently displayed window that contains the cursor location. Thus it can determine which point in which image was intended. The window facility lets the user operate as though a collection of photos were on a desk. from which one can be pulled and placed on top for examination.

or comparison with another, while fragments of the remainder remain visible.

When the user asks for a picture or a portion of a picture to be displayed in a particular window, the server determines whether to expand (by replicating pixels) or reduce (by sampling pixels) the image to fit the window. When the image is thus displayed, the picture source file is associated with the window, so that the user may subsequently have the window redisplayed, either naming it, or pointing to it. The user may also, for example, have an enlarged portion of any window displayed, pointing with the cursor to a destination window, and to the feature to be enlarged.

Each display window has associated with it two coordinate frames in addition to the absolute coordinate frame of the display hardware: One is a logical-coordinate frame, the other a world-coordinate frame. The logical coordinates relate a point in the window to the digitized picture from which the displayed image was derived. Thus, the logical coordinates of a feature on the screen will be the same, no matter where the display window is, or whether the image it presents is an enlargement or a reduction of the original. The logical coordinates are defined when the command is given to display a particular picture in a particular window. The world-coordinate frame is the UTM grid. When a digitized picture has been calibrated and the resulting transformation information filed, the appropriate transform can be read in and associated with the display window, so that locations of features in the image may be given to the user in UTM grid locations with elevations. Since the imaging process loses one dimension, there is a one-dimensional ambiguity in specifying the world location of the feature. This ambiguity is removed either by asking the user to specify elevation or by retreiving the elevation automatically (when it is available) from the terrain data base.

The display server, through the basic capabilities described above, makes it possible for the user to point at locations in the image and be informed of their world location. Alternatively, the Hawkeye system can

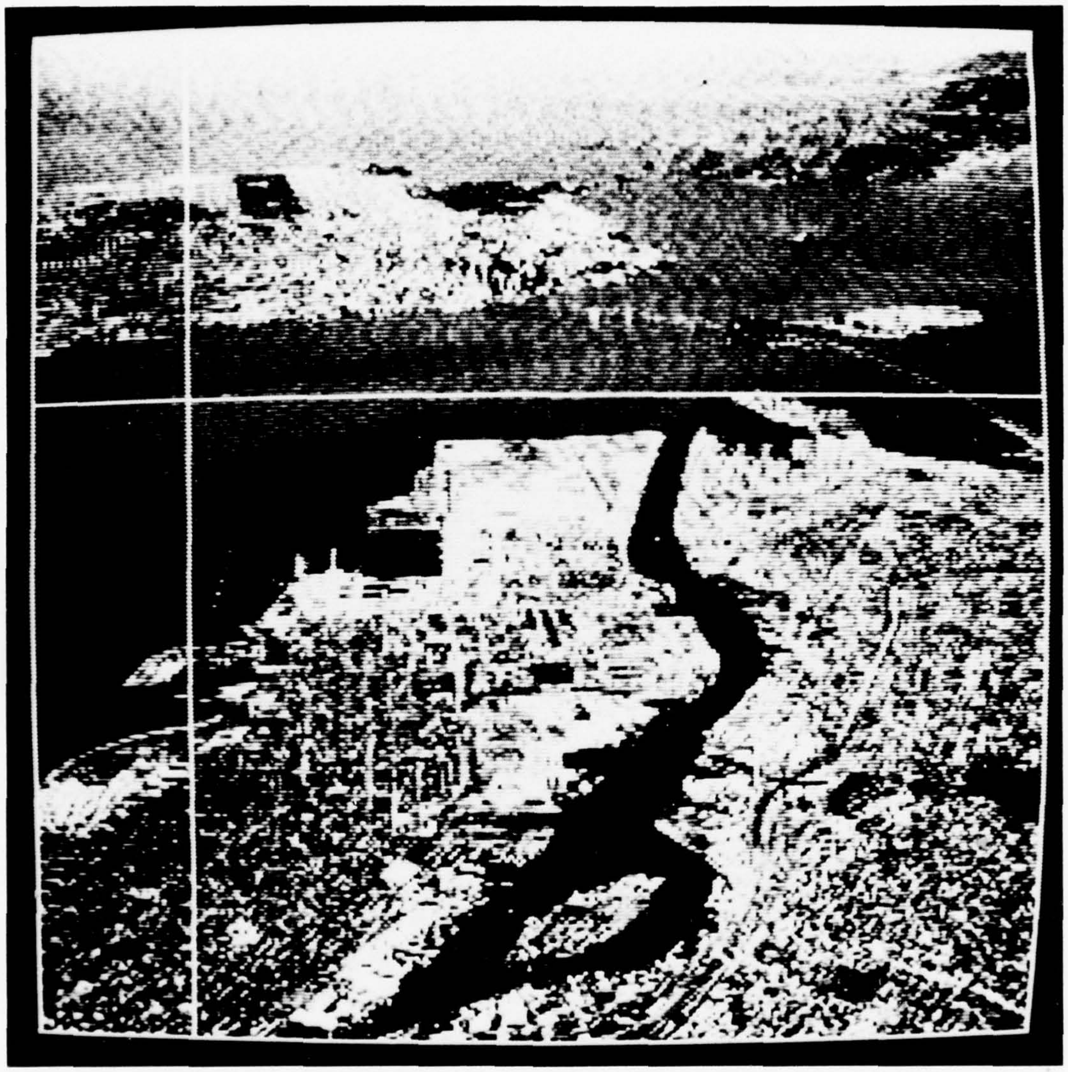

FIGURE 3 DEFINING A DISPLAY WINDOW

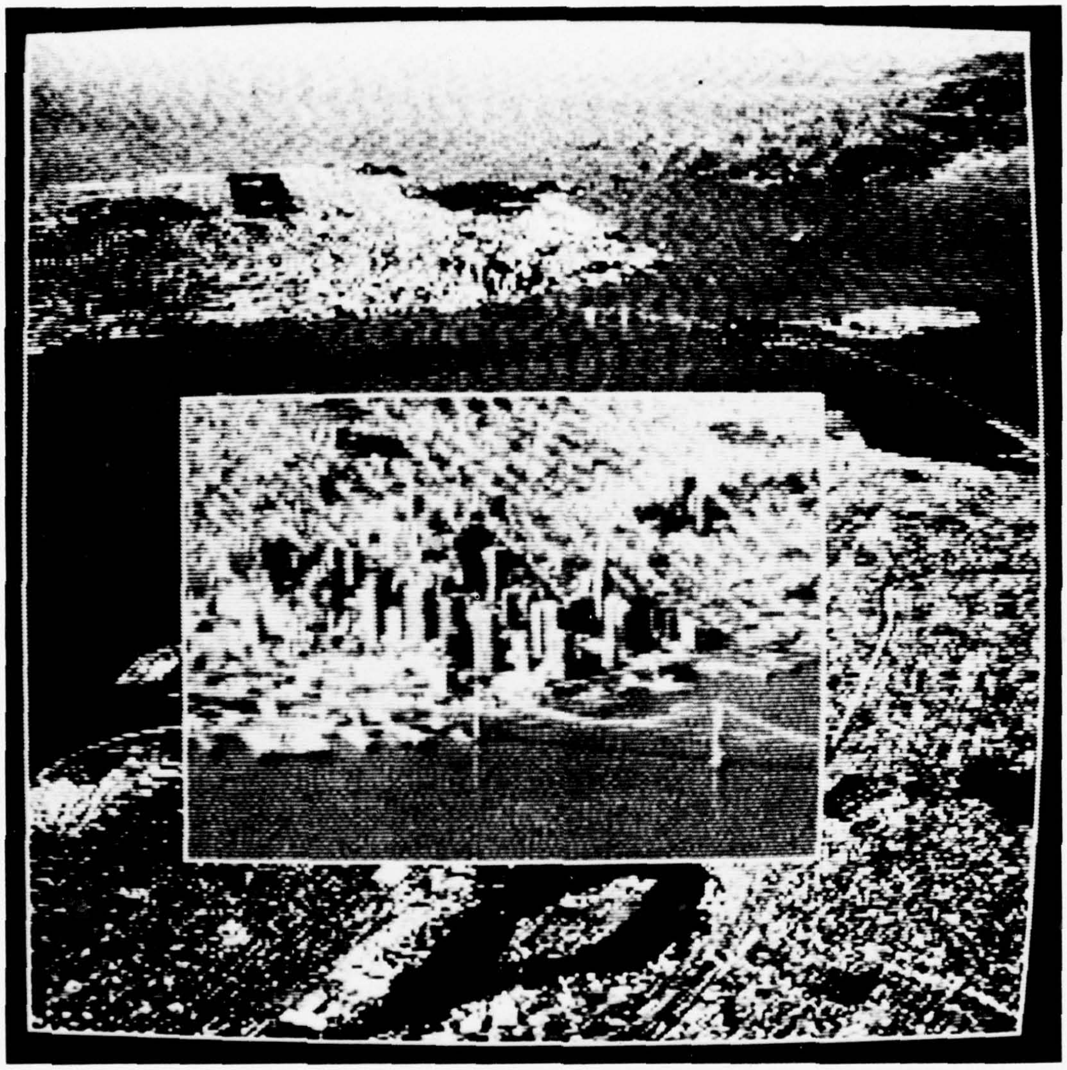

FIGURE 4 AN ENLARGED INSET

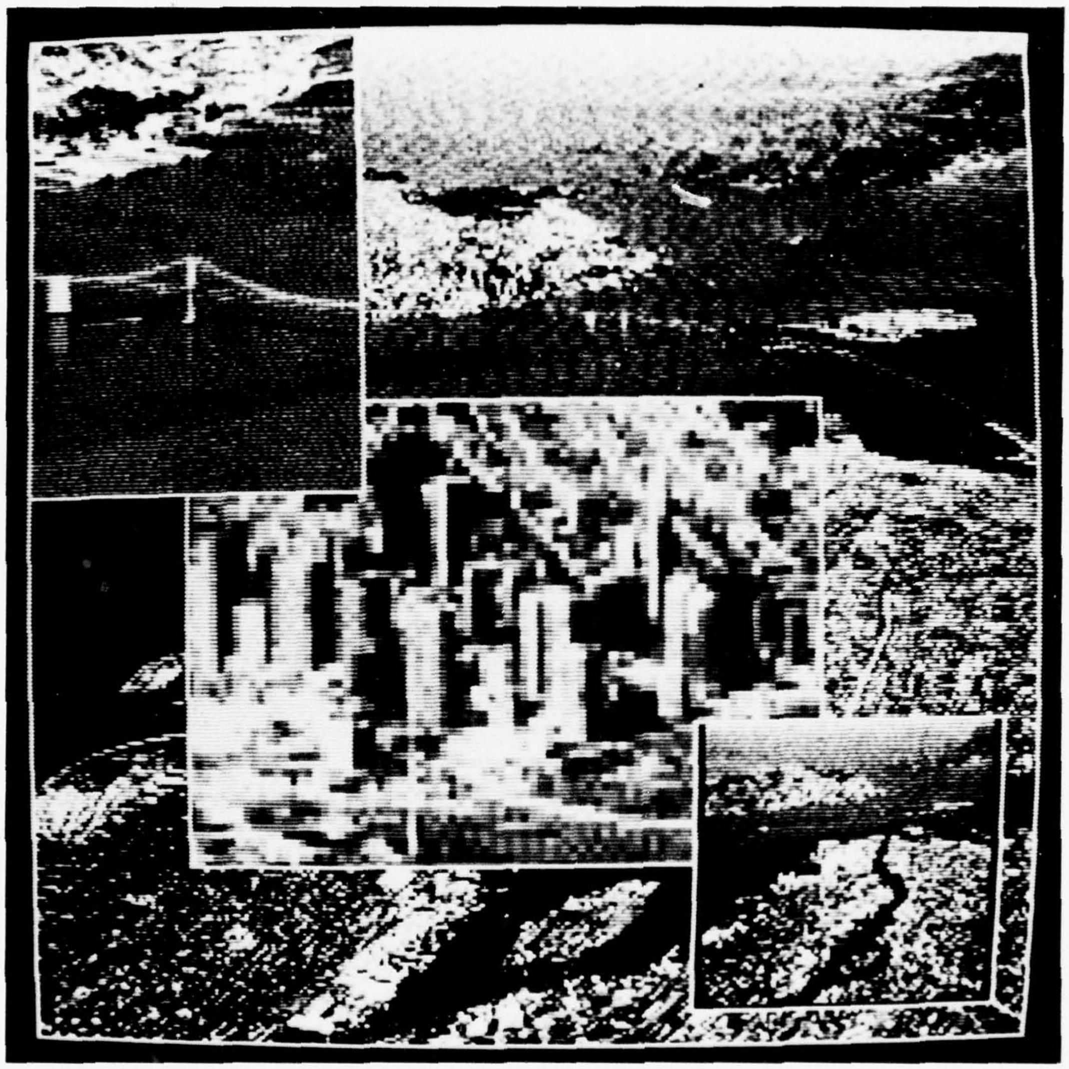

FIGURE 5 MULTIPLE DISPLAY WINDOWS

automaticallv identify the feature bv inauiring of the data base the known features near that location. Converselv. since the relation between world and displaved image is known for each window. it is possible to determine where anv feature in the map data base should appear in the image. and hence to overlay it on the displav. Having the svstem respond to the twin queries: What road is that? and Where is road x? can be verv useful to a photo interpreter.

C. Graphics Tablet Server

Two graphics tablets are used with the Hawkeye svstem: The larger one has a 3- by 4-foot digitization area: The other has an 11-inch square digitization area. A single cursor may be used with either tablet to encode the location of a point to 0.001 inch resolution (with 0.005-inch accuracv). The tablets permit the photo interpreter to use documents such as standard printed maps or original photos in conjunction with digital data.

The graphics tablet server is essentiallv a restricted version of the display server. The user can fasten several documents to the large tablet. (e.g.. a conventional topographic map and several aerial photos) and define a tablet window (a rectangular region of tablet surface) for each. When the user indicates a point with the cursor. the svstem can determine which window is indicated. and hence which document is being used. Tablet windows. like display windows. mav overlap. For determining which window is indicated. however. the internal representation orders windows bv recencv of creation. rather than recencv of use. since newer documents are usuallv fastened on top of older ones.

The system can indicate a point on a document bv printing the tablet coordinates of the point. The user mav then move the cursor until the numerical readout on the tablet control unit displavs the specified location. Thus. the svstem can use the tablet to respond to such questions as: Where is X on this map?.

Each tablet window, like a display window, may have logical- and world-coordinate frames associated with it. The logical coordinate transformation must be determined for each tablet window individually by indicating calibration points with the cursor and typing the corresponding logical coordinates. For example, a tablet window enclosing a topographic map can be calibrated so that points indicated by the user are converted directly to UTM coordinates. Documents that do not represent a simple orthogonal projection of the world, such as photos, can also be calibrated in terms of camera parameters. It is then possible, using hardcopy photos, to obtain world locations of indicated features, perform mensuration tasks, and to determine image features corresponding to a given set of coordinates in much the same way as with digitized images on the display.

Since we handle tablet windows and display windows in a consistent way, it is possible to mix digitized images and printed documents freely. For example, a new digitized image may be calibrated by pointing to corresponding features on the display and on a topographic map.

Conventional printed maps can be used with Hawkeye for two purposes: to supplement or substitute for the digital map in areas for which maps in digital form are not yet available and to input information to the map data base. For the latter purpose, we have written programs that allow the user to trace features directly from the topographic map, have them displayed for verification and possible editing, and then incorporated into the map data base. Elevations must currently be entered manually when tracing from a map. However, we should soon be able to obtain this third dimension automatically from digital terrain data, when they become available.

D. Calibration Server

The calibration server provides routines to compute the transformations used in coordinate conversions, such as between UTM map coordinates and digitized picture coordinates. It also provides the basic mensuration functions, such as height or distance determination, which rely on the coordinate conversions.

The coordinate systems that are commonly used in Hawkeye include graphics tablet coordinates, UTM coordinates, relative UTM coordinates, film plane coordinates, digitized picture coordinates, and display coordinates. Coordinates are represented as homogeneous vectors, and the transformations that map between them are represented as 4x4 matrices. This representation is convenient because it can be used to represent both linear and perspective transformations.

The basic calibration routine computes a best estimate of the transformation, in a least-squares sense, between two lists of corresponding coordinates. This optimization routine is used in conjunction with a variety of other modules concerned with determining corresponding calibration points in different coordinate systems. To facilitate experimentation, an interactive routine is available that allows the user to specify the source of each list of points (graphics tablet, display, or file) and the type of transformation (digitization, perspective, or collineation), and (if appropriate) to indicate control points on the designated input devices. The interactive routine then calls the optimizer to compute an estimate of the transformation and displays the residuals. If the user is not satisfied with the precision, the two lists of points can be edited and the computation tried again.

The mensuration primitives allow the user to measure distances in terms of any of the above coordinate systems, e.g., the length of a ship or the height of building. The routines are general enough so that the user can indicate each point in a different window (graphics tablet or display). For example, it is possible to measure the distance between a point on a map currently on the graphics tablet and a point in a digitized picture currently being displayed.

E. Map Data Base Server

The data base server is the means of access to the map data base for those system modules that are not required to have detailed knowledge of its structure or implementation. This server contains access routines for answering a variety of standard queries about specific data and the general format of the data base; for example: What is at (x,y,z)? What is the closest road to (x,y,z)? What roads are contained in the area bounded by ...? Where is Oakland Mole? What is the <attribute> of <object>? What is known about <object>?

The map data base contains three-dimensional descriptions of cartographic and cultural features, including coastlines, major roads, lakes, bridges, airfield runways, oil storage tanks, and harbor lights. In addition, the map contains a partial taxonomy of world entities, with relevant general semantics, information about available imagery, and descriptions of data structures used by the system. The information about imagery includes file name, calibration data, and geographic area covered and can be used in selecting appropriate pictures for specific tasks. The descriptions of the data structures enable the system automatically to construct (instantiate) new entities of the correct structure for inclusion in the data base.

The map data base is a disk-based semantic net data structure that can contain realistic quantities of data represented in a way that permits efficient access. Entities are represented by LISP atoms (e.g., English words); information associated with the entity is stored in a property list format. Relationships to other entities are also stored on the property lists, thus establishing a network structure in the data base. When information concerning a particular entity is sought, the property list is retrieved from disk and established in core. A "paging" scheme limits the amount of data in core (to, say, 1000 entities) and writes entities back out to disk, if necessary, the least recently used ones first [2]. Retrieval of the information is by means of a hash table on disk, which means that access time is constant and independent of data base size. The geometric data are indexed (the index structure is part of the data base) via K-D trees [4], one tree for each class of entity sought, to enable fast retrieval of information relevant to a particular area. Further details of data base implementation are to be found in Appendix A, and examples of semantic organization are given in [2].

We are setting up a map of the San Francisco Bay Area, containing major features, coastlines, bridges, and highways. Figure 6 is a portion of a U.S. Geological Survey (USGS) map of the area; Figure 7 shows the portion of the map currently in the data base. Figure 8 shows part of the map at higher resolution. The map consists of about 4000 points, plus various semantic relationships, totaling about three-quarters of a million bytes of disk storage. (Access to a particular item of information takes less than 1 ms if it is paged in, and 15 to 30 ms plus disc access time if it must be read in). The map information is entered by manually tracing features on a USGS map using a digitizing table: Map data in digital form are not readily available; because the problem of digitizing printed maps has rather different constraints from the problem of making maps from photographs, we could not exploit our guided tracing techniques.

F. Terrain Data Base Server

Information about the detailed topography of the San Francisco Bay Area is available as an array of elevations over a grid with 200-ft spacing between samples. The amount of data involved is very large (several million bytes in its original format); even so, it is necessary to interpolate to estimate elevation for an arbitrary location.

The terrain-data-base server performs two main functions. The most basic function simply returns the elevation of a location specified in UTM grid coordinates. To do so requires performing a local interpolation (currently bilinear).

The second function is, given two locations in three-dimensional coordinates (a viewpoint and a world point) that specify a line of sight, to determine the point on the ground that will be seen looking

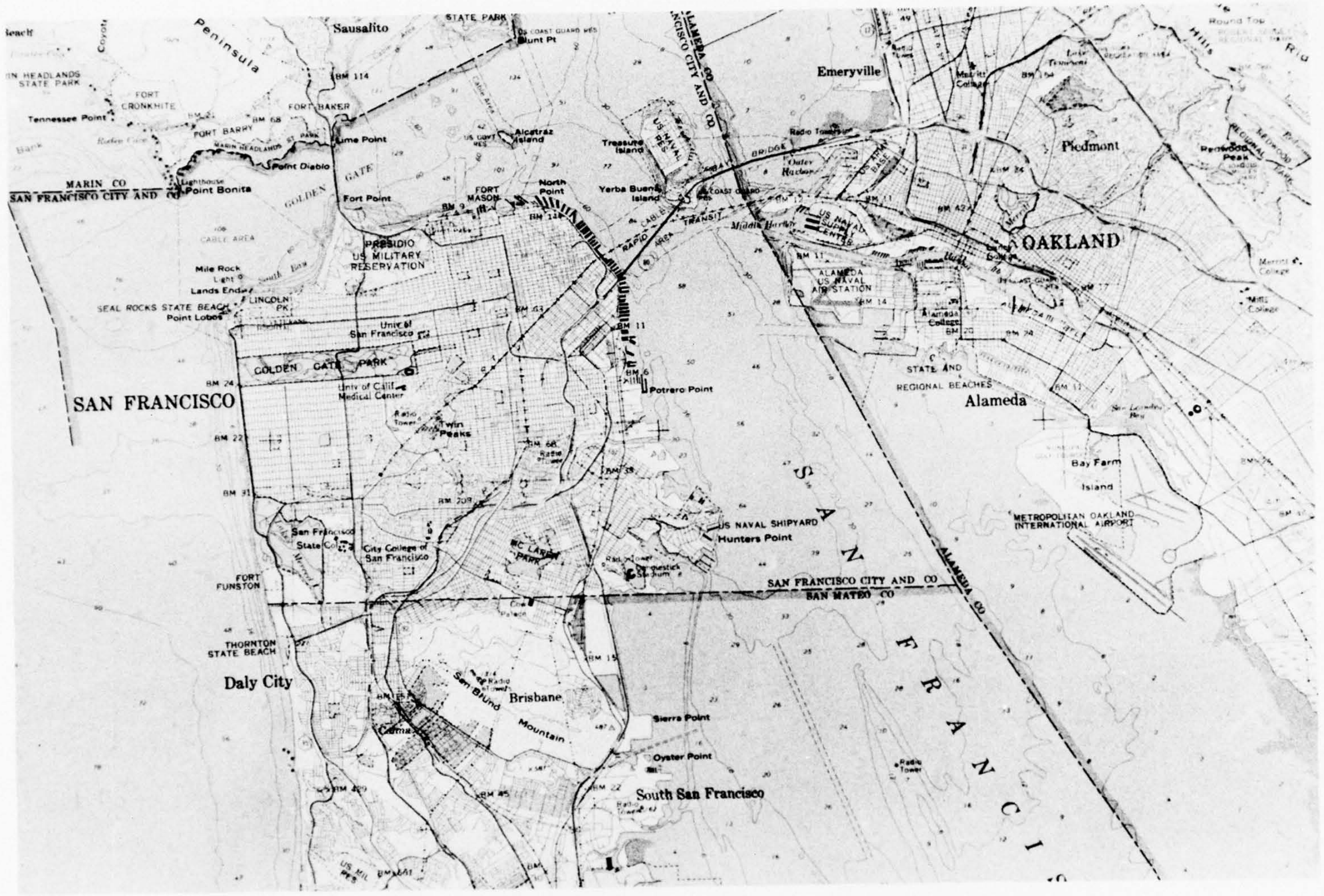

FIGURE 6   USGS MAP OF THE SAN FRANCISCO BAY AREA

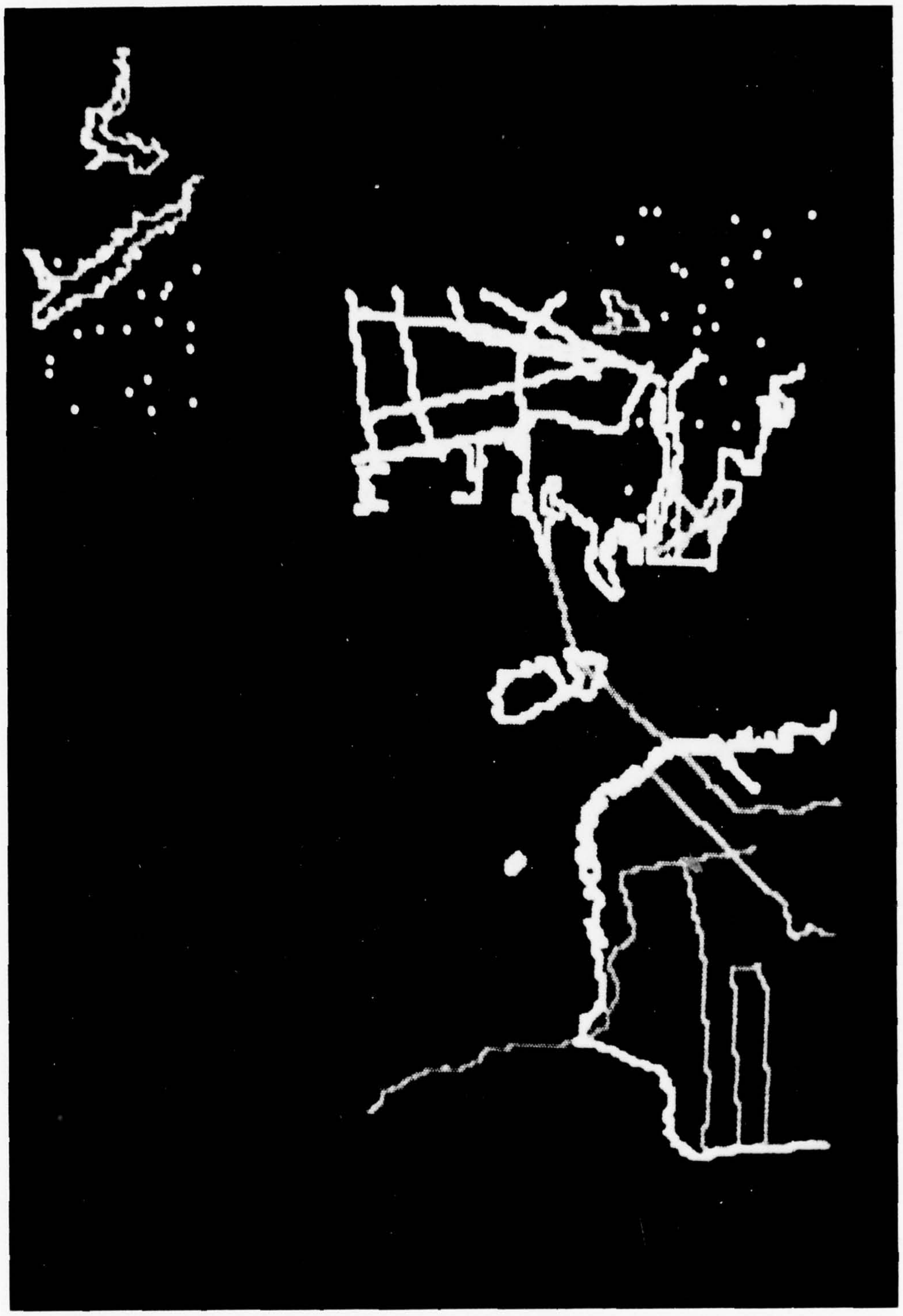

FIGURE 7   DISPLAY OF THE DIGITAL MAP DATA BASE

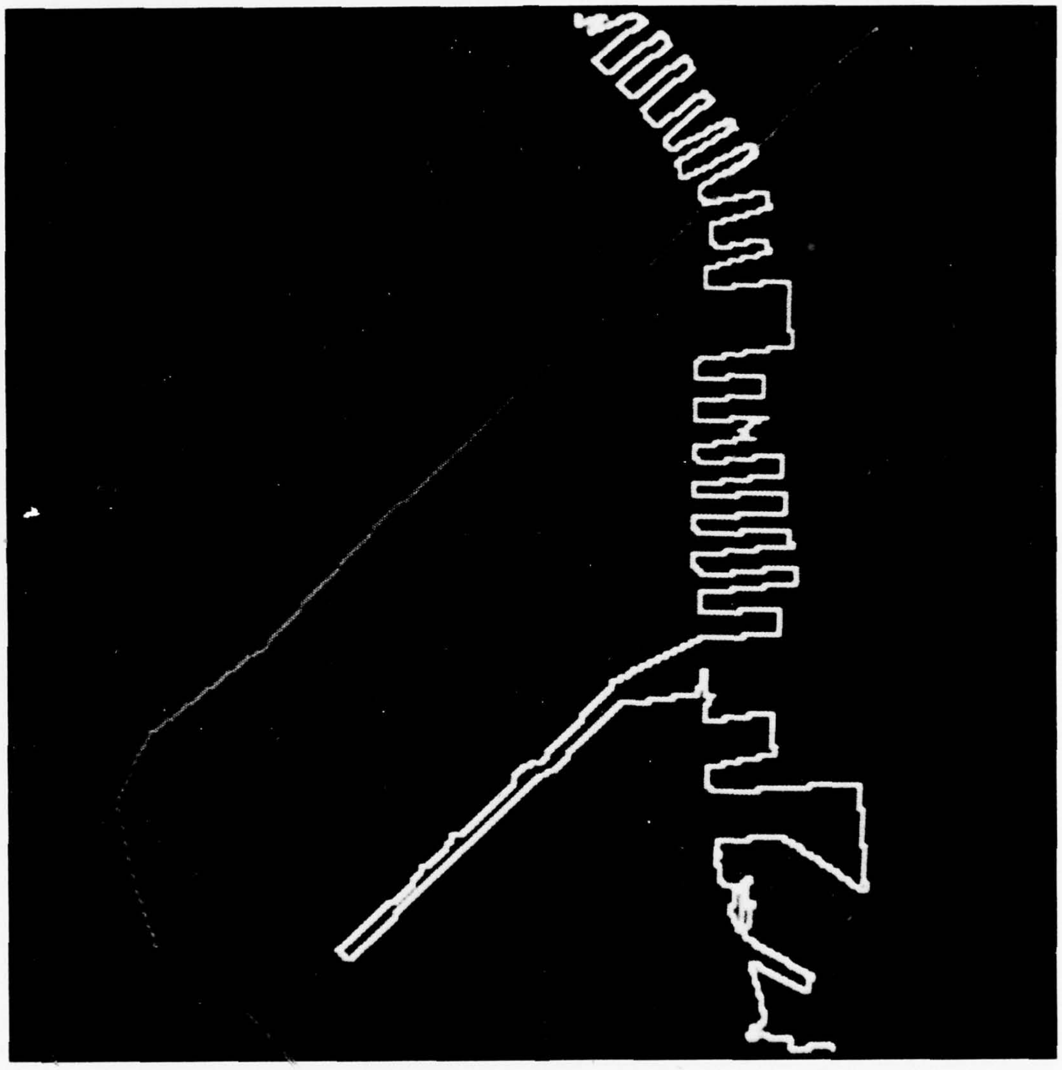

FIGURE 8 PART OF THE MAP AT HIGHER RESOLUTION

along that line of sight. This function is important, because it enables estimation of the third dimension that is lost when an image is formed: Each point in a calibrated image specifies only a line of sight, and the terrain information resolves the implied ambiguity.

The terrain-data-base server can be used whenever locations or distances are estimated from an image. For example, when measuring the height of a vertical structure by pointing to its top and bottom, the terrain data specify the location of the bottom, and thereby set the scale of the structure.

The utility of terrain data is limited by their accuracy. We obtained our Bay Area data from the National Cartographic Information Center, Reston. The data base was originally created by tracing contours on a 1:250,000 scale map and then interpolating between contours to estimate elevations at grid points. There are thus three classes of errors: those existing in the original topographic map, those arising during tracing, and those resulting from interpolation. We have already become aware of peculiarities in the data resulting from the interpolation algorithm, but we are not yet able to assess their seriousness. We have, however, discovered gross errors apparently owing to faulty copying of the digital data before distribution. For these reasons, the terrain information used in the Hawkeye system is considerably less accurate than it need be. An operational photo interpretation certainly could have access to more precise and more detailed information.

G. User Interface

1. Facilities

When a system becomes large and complex, it may become difficult for the user to remember the command format, qualifications, and consequences for all its components, particularly if some are infrequently used. The situation is even worse if the system may be used by those who only encounter it occasionally, rather than spending their working day with it. During system development, when several people are experimenting with and modifying various pieces, no individual has complete knowledge of all aspects of the system.

To facilitate interaction with the Hawkeye system, we have implemented a user interface module that provides two main facilities: natural language communication and a help facility. Natural language (more accurately, a subset of it) gives the user considerable flexibility in structuring queries or commands so that he or she is not restricted by any difficulty in recalling a formal syntax. The help facility enables the user to find out the range of capabilities and data possessed by the system. The user interface also includes task queueing and reporting facilities to simulate a photo interpreting environment, such as that provided by the PACER system.

2. Natural Language Communication

The user interface is implemented with LIFER, a proprietary language definition and parsing system written in LISP and developed at SRI by Hendrix [5]. LIFER interfaces have been designed for several large AI programs, including the ACCAT test bed supported by ARPA [6]. LIFER makes it particularly easy to achieve dialogs ıt a limited domain, facilitated by such features as acceptance of elliptical input and the capability for the user to expand the grammar incrementally as deficiencies are discovered.

LIFER uses an augmented transition net grammar whose symbols correspond to semantic as well as syntactic entities. The language designer provides a collection of rules, each of which consists of a pattern and a response expression. For example:

```
   PATTERN.DEFINE( (WHAT IS THE DATE) (PRINT (DATE)) )
```

will cause the expression (PRINT (DATE)) to be evaluated when the user types: What is the date?.

The designer can also define patterns that use metasymbols to stand for subexpressions. For example:

```
   PATTERN.DEFINE( (WHERE IS <PLACE>)
                        (PRINT (LOCATION <PLACE>)) )
```

will print the location of whatever is bound to the metasymbol <PLACE> as a result of the pattern matching the input. If the user types: Where is Oakland?, and <PLACE> matches "Oakland," then the expression (PRINT (LOCATION 'OAKLAND)) is evaluated.

A metasymbol can be defined by one, or more, of an explicit set, a predicate, or a subgrammar. A set can be defined by, for example,

```
        MAKE.SET( <PLACE> (SRI OAKLAND ALAMEDA) ),
```

which makes SRI, OAKLAND, and ALAMEDA acceptable places.

A predicate is similarly defined:

```
        MAKE.PREDICATE( <PLACE> PLACE.TESTER ),
```

where PLACE.TESTER is the name of a function that can decide whether to accept a given word as a place name.

Subgrammars are defined in terms of patterns and response expressions, using PATTERN.DEFINE:

```
        PATTERN.DEFINE( <PLACE>
                        (THE CITY HALL OF <CITY>)
                        (CITY.HALL <CITY>))
```

The above definitions would accept: Where is the city hall of Sacramento? and return the result of evaluating (LOCATION (CITY.HALL 'SACRAMENTO)), assuming that <CITY> were also defined.

Multiple patterns may be defined for the grammar or a subgrammar. Each pattern definition is added as an alternative to (not a replacement for) what already exists. Subgrammars may be recursive or mutually recursive, so that complex phrases, such as: the nearest road to the tallest building visible in the current photo." may readily be handled.

Note that the subgrammars <PLACE> and <CITY> in the examples above have semantic as well as syntactic content. In a conventional grammar, both might have been replaced by <NOUN.PHRASE>, but this would have two significant disadvantages: First, the grammar would accept many more nonsense sentences because it would not be able to make fine semantic distinctions (recall "colorless green ideas"). Second, it would not be able to make different responses to semantically different but syntactically similar input. In addition to embedding semantics in the patterns, LIFER permits semantic checking during response evaluation. The designer can define response functions to return an error signal if they encounter a problem. Control then reverts back to the parser, which looks for an alternative interpretation of the input.

The response expression may be defined either to return a value directly (in which case semantic checking may be performed immediately) or to return a LISP S-expression to be evaluated later. The latter is preferable if evaluation is expensive or entails output, since the same expression can be encountered several times during the search for an acceptable parse. In Hawkeye, many response expressions can be evaluated immediately, but when a data-base query is concerned it is more efficient to construct a complex S-expression that is transmitted to the data base server and evaluated as a single entity. Thus, LIFER might be used to translate a complex English sentence, such as Where is the city hall of the nearest city to the smallest lake in California?, into a complex LISP expression, such as

```
(LOCATION
    (CITY.HALL
        (NEAREST
            (LEAST AREA (IN.REGION CALIFORNIA LAKES ) )
                CITIES ) ) ).
```

LIFER possesses two features that make interaction easier for the user. The ellipsis feature enables the user to type in only a fragment of a sentence. If it cannot make sense of the fragment as an independent utterance, LIFER tries to fit the fragment into the context of the previous input. For example, if the user asks: Where is the city hall of Oakland?, and then asks: of Concord?, LIFER assumes he meant: Where is the city hall of Concord? and responds appropriately.

The paraphrase feature allows the user to make his own extensions to the grammar, without having to understand the mechanisms or formats involved. For example, he can type: Let 'Where is Oakland citv hall?' be a paraphrase of Where is the city hall of Oakland? LIFER analvzes both example sentences, compares them. and decides that the appropriate thing to do is to extend the definition of <PLACE> by adding the pattern (THE <CITY> CITY HALL) with the response expression (CITY.HALL <CITY>) . The grammar will subsequentlv accept: Where is Concord city hall?

There are many commonly used contractions that people use in conversation. and often in written communication. For example. "where's" for "where is?", and "I'm" for "I am". Many of these could be readily catered for with the existing LIFER machinery. but some would cause problems. For example. "can't" expands to "cannot" and "can" should really be associated with one phrase. and the qualifier "not" with another. To permit this, we added a facility to LIFER to perform expansion of recognized contractions before the input is passed to the parser. The same facility also deals with punctuation. making sure that it is properly separated from the words, discarding ignorable punctuation (such as question marks or periods). It also permits the user or designer to define synonyms. such as "c.h" for "city hall."

The amount of work involved in designing the grammar for the user interface was comparatively small -- about a man-month. The grammar produced covers many different ways of expressing commands that exercise most of the system's capabilities. Our experience with the system has shown that it is useful to be able to give commands in English. once the grammar has been specified sufficiently well to cover the preferred constructs used by a small group of users. A user can be relatively free in expressing himself and the system will understand. We do not believe, however, that we yet have a grammar complete enough to understand the full range of constructs commonly used by most of the population. At this point. it is not clear whether the grammar could be extended sufficiently without requiring significant reorganization of at least some of its parts. It appears that in situations involving naive users and restricted domains, an interface of the LIFER variety is well worth while. For richer domains with complex structured data and a wide range of capabilities, the interface may become disproportionately large and complex. The Hawkeye system--and its data base in particular--may be a domain that is already too rich for a simple grammar covering most constructs. Natural language may be unnecessary for an experienced user who can spare the time to learn an extensive formal language.

3. Help Facility

No matter how experienced A user--no matter how experienced--dealing with a complex system will occasionally require assistance, perhaps in the form of a reference manual. We have therefore provided a simple help facility for the Hawkeye system.

Although a user in need of help may be able to frame a very specific question, there are usually few constraints on what might be asked. Rather than try to cover the potentially enormous range of questions that might be posed, we decided to use the English interface only to invoke the help facility. The facility itself operates by presenting the user with a chunk of information and giving him a small set of choices for getting further information.

The help information is organized hierarchically, with chunks of information associated with nodes in a tree of contexts. A context also has a heading or name that reflects its position in the hierarchy. The topmost node, for example, has the title "Hawkeye." Each context has various types of information associated with it:

* Description--a brief (one page at most) description.
* Subtopics--a list of the more specific contexts immediately below this context.
* Instructions--how to accomplish a particular goal, in as much detail as appropriate for this context level.
* References--other documentary sources of information.
* Parameters--(where appropriate)
* Relations--to other parts of the system.
* People--for assistance or complaints.

Thus the help context tree has several types of information associated with each context, and the tree descends only from the subtopics.

The top-level node, with the context "Hawkeye," has information under the headings "Description" and "Subtopics". "Hawkeye : description" has a brief (one paragraph) overview of the Hawkeye system. "Hawkeye : subtopics" reveals that there is more detailed information about hardware, the data base, application programs, the help facility, people, documentation, and the task queue. Each subtopic has its own brief description, and most also have their own subtopics.

A user in a particular help context (corresponding to a node on the information tree), can enter an information type (e.g., description, subtopics, instructions) and have the appropriate text printed. For example, requesting subtopics information results in presentation of a list of subtopics for which more detailed information exists. The user can then type a subtopic name (or its position number in the list) and descend to the more detailed context. Alternatively, the user can ascend to the context immediately above the current context, by typing "Up," or to the topmost context of the tree, by typing "Top." The user can cycle through the information types for the current context, by typing "More," and cycle backward by typing "Back." To redisplay the text for the current context and type, the user types "Print," and to find out the current context he types "Context." The latter command might print, for example,

```
   HAWKEYE / DATABASE / DIGITIZED IMAGES : DESCRIPTION
```

The user can obtain a complete index of the context tree by typing "Index," and will then see a formatted and alphabetized list of the topics, subtopics, subsubtopics, and so forth:

```
HAWKEYE
        APPLICATION PROGRAMS
                ASK
                CORRESPONDENCE
                IMAGE ANALYSIS
                ROAD TRACER
                SHIP MONITOR
                TELL
        DATABASE
                ...
```

```
DOCUMENTATION
        ...
HARDWARE
        GRAPHICS TABLE
        RAMTEK COLOR DISPLAY
        TRACK BALL
        ...
HELP FACILITY
PEOPLE
TASK QUEUE.
```

The user can jump directly to a particular context simply by typing "Search" followed by the topic name. He or she will then find himself in the most general context concerning that topic. A user who invokes the help facility by simply typing "Help" is placed in the context that describes the help facility itself and information about the basic help commands is printed.

4. Tasking and Reporting

Tasking and reporting are important parts of an overall photo interpretation operation. Hawkeye has facilities representing both. A task queue contains requests expressed in English, as they might be submitted by intelligence analysts. The user can ask the system whether there are any tasks in the queue, for example: Are there any more requests?. Even if the user does not pose the question, the system will automatically check at intervals (currently about five minutes) and warn the user if the queue is not empty.

The user can ask: What is the next task?, and the next request is read in from the queue and printed. The user presumably then tries to comply with the request. When finished, the user can report findings directly in text by "Report 'The latest coverage indicates .... ' ", as well as by putting information into the data base. He or she then tells Hawkeye "I've finished that job," or something similar, and asks for the next task.

If a task is sufficiently straightforward for Hawkeye to perform automatically, the user can ask Hawkeye to "do it automatically." Hawkeye then uses the natural language machinery to interpret the request and execute it. The user can also say "do all the tasks automatically," and the system will attempt to perform each task in turn, until the queue is empty. If a request is encountered that Hawkeye is unable to parse or execute, it informs the user, prints the request, and stops so that the request may be fulfilled interactively.

III CONCLUSIONS

A. <u>The Hawkeye System and Its Components</u>

The Hawkeye system, at its present stage of development, contains many utility service routines that are fundamental to almost all nontrivial photo interpretation tasks. The most important of these are the data base for specific and general knowledge, the image-processing routines for realistic size and resolution of imagery, the coordinate transformation and calibration routines for relating images to the real world, the interprocess communication routines for organizing large, resource sharing systems, and the routines for communicating with a human user through multiple media.

Some of the components are now sufficiently self-contained or well-defined that they can be extracted and used elsewhere. The basic data-base routines, for example, enable construction and manipulation of very large semantic nets, not restricted to representing maps. Also, the inter-process communication routines are of general use in building large, complex programs. These routines will soon be employed by other projects at SRI, and are freely available to other members of the image-understanding community.

Our work in the initial stages of this project has been mainly exploratory: we believed that many cartographic and photo interpretive tasks currently performed manually could be at least partially automated using an image understanding approach. We looked at examples of four common tasks that are labor intensive, tedious, and bottlenecks to productivity: tracing, counting, measuring, and monitoring. Techniques were developed to automate (or at least greatly facilitiate) each of the selected tasks. In several cases, performance on our limited data set has been sufficiently good to warrant more extensive testing and evaluation. We are, for example, planning to transfer the interactive road tracing routines to the U.S. Army Engineering Topographic Laboratories.

The techniques used for ship monitoring and boxcar counting demonstrated the use of a digital map to guide automatic interpretation of imagery: a key concept was that relatively simple analysis techniques could be used once an appropriate area of the image and its context were defined. The usefulness of this approach is somewhat limited by the infeasibility of developing specialized programs for every different monitoring and counting application. A photo interpreter must count boxcars, oil derricks, planes, and many other articles with equal dexterity in a wide range of contexts. Although ways can be found in many cases to automate such tasks, using the map for guidance, clearly no single image-processing algorithm will suffice. It is therefore necessary to develop systems that can be rapidly taught to recognize new objects by interactively designating examples in an image, along the lines of [7] and [8]. Before such systems are possible, however, it is first necessary to to determine appropriate analysis techniques, and knowledge required to guide them, by investigating a range of example tasks. We have already made preliminary investigations of several different tasks, but a more detailed study of a broad range of situations is now required.

Another serious problem concerns the reliability of the system. Using simple techniques and limited amounts of knowledge renders the system easily fooled; the ship monitor, for example, does not invoke much knowledge of ships and thus cannot always distinguish between a single large ship and a cluster of smaller craft. In order to attain a high level of autonomy and robustness over a wide range of situations, it is necessary to provide the system with a great deal of knowledge and the routines for exploiting it. To make progress in this area, it is necessary to select some class of tasks and a domain that are sufficiently constrained that it appears possible to codify most of the required knowledge, and are still worth pursuing on the grounds of military relevance. We are now directing this project toward specialization in such a domain.

B. New Directions: Road Monitoring

Hawkeye demonstrated the feasibility of using knowledge about maps and imaging to automate a variety of representative photo interpretation tasks. With this knowledge, adequate performance was achieved in straightforward cases, but the system was easily misled by contingencies that it did not know about, for example, clouds. Substantially more world knowledge and a greater range of capabilities that use the knowledge are necessary to approach human performance. The Hawkeye system framework provides a suitable foundation for integrating the knowledge and capabilities into an expert system. In the next stage of our research, we plan to develop a system with considerable expertise in a specialized task area.

The task we have selected is that of monitoring traffic on roads. More specifically, given a sequence of reconnaissance images of a region under surveillance, possibly taken under adverse viewing conditions, the system will first locate sections of known roads visible in the images, locate anomalous regions on the roads whose size, shape, velocity, and other characteristics are consistent with those of vehicles, and then perform a detailed scene analysis in the vicinity of the anomalies in order to identify specific vehicle types.

In order to attain the level of performance for which we aim, the system will require expertise concerning roads and vehicles, including knowledge of a wide variety of situations and events, such as obscuration of roads by trees or clouds, the visual effects of snow and rain, the behavior of roads at intersections, mountains, tunnels, and so forth. The system will also require knowledge of its repertoire of resources, their abilities and limitations, and how to evaluate its own performance.

Appendix A

THE MAP DATA-BASE SERVER

Appendix A
THE MAP DATA-BASE SERVER

The map data base is the primary source of knowledge for all task specialists in Hawkeye. In this Appendix, we discuss the implementation details of the map data base. We begin with the primitive file storage system and then discuss the semantic data base and the overlaid geometrical data base.

1. File/Buffer System

The file data in the IU data base is stored in extended atom-property list format. That is, the items stored in the data base are LISP literal atoms and their associated values are lists in standard LISP property list format. The file is held in text format, since the LISP reading routines are very efficient at converting text to internal data structures.

In order to retrieve an item value from the file, a (hashed) file address is computed for the atomic token, and the associated value is retrieved from that location in the file and stored in a core-resident buffer, called DBBUFFER. Subsequent accesses to the same token will retrieve the value stored in DBBUFFER. The buffer typically has a capacity of 500 to 1000 tokens.

When DBBUFFER is full and another slot is required, the system empties the least recently used (LRU) slot. If the LRU item has been modified, it is written onto the end of the file and its hash entry updated to point to the current data; if not, it is simply discarded. The new data element is inserted into the now-vacant slot, which is then promoted to be the most recently used slot. Any access to a data element causes its slot to be promoted to the front of the buffer, maintaining the ordering by recency of use.

Any token currently in DBBUFFER has on its real property list a pointer to its slot in the buffer. This property is removed when it is deleted from the buffer, which allows the system to tell immediately whether an item is in core.

The hashed file has two parts, the first of which is the hash table consisting of a number of "buckets." Each bucket contains either the byte address of a piece of data stored in the second part of the file or, if the bucket is empty, zero. The data stored in the second portion of the file consist of token names and their values.

The process of retrieving the value of a target token uses a double-hash process with open probing. A hash token is first created from the atom name. The value of this hash token modulo the first hash divisor, HASH1.DIVISOR, is used as the first hash address. If the bucket at that address is empty, the target is not in the file and NIL is returned. If the atom stored at the location pointed to by the bucket matches the target, its associated value is returned; otherwise, there is a collision, and a new address must be computed. A second hash address is computed by adding the value of the first hash address to the value of the original hash token modulo HASH2.DIVISOR (a number equal to the length of the hash table), and the process is repeated. If this also results in a conflict, the system resorts to open probing (i.e., it proceeds sequentially, beginning at the second hash address until it either finds the target item, or an empty slot).

This type of hash addressing does not allow for the easy deletion of items from the table. In addition, since an entry is changed by writing the new value on the end of the file, the data base file tends to become cluttered. To alleviate these problems, a routine is provided for "tidying" the file. This program creates a new file by going through the hash table of the old file, retrieving all items with nonatomic values, and inserting them into the new file. Therefore, an item whose value has been set to NIL will be deleted when the file is cleaned up.

2. Semantic Data Base

The semantic data base is a network structure, the nodes of which are atoms stored on the data base file. The primary ordering principle imposed on the net is the superset/subset relation. The elements of a set are either enumerated or are implicitly determined by a special predicate associated with the set. Canonical set elements are described by a "delineator," which specifies the arcs that describe relationships between the elements of the delineated set and other entities in the data base.

Two major classes of information are stored in the semantic data base: image-related data and map (or world) data. Image data include everything known about photographs, digitized images, and calibration files. The world information describes generically a set of cultural and cartographic features and various instances of these features.

Pertinent image data consist of the date and time a photograph was taken, the approximate camera parameters, the area covered (actually the world coordinates -- UTM -- of the corners of the photograph), and, (if calibrated) the camera coordinates. Calibration files for images allow the transformation between image-plane coordinates (inches), digitized image coordinates (pixels), and map coordinates (UTM). The data base records the file names of digitized images (and subimages) and the names of the calibration files. This data allows programs for locating map objects in images to retrieve the necessary information.

The geometric data structure (described below) allows the storage and retrieval of images based on their coverage. Thus, the system can answer such questions as: What images contain the Oakland Mole? By consulting date information, more complex queries such as, What is the most recent, calibrated photograph of the Oakland Mole? can be answered.

The map data in which we are interested have two main aspects: the semantic description of the object itself and a topological/geometrical description of the location, extent, and form of the object. Semantic data stored about a road, for example, would include features such as the number of lanes, the type of surface, and the classification (i.e., freeway, avenue). Geometric/topological data would specify the fact that the road is a linear feature of a certain length, connecting various places, and that entry and egress are usually possible only at certain defined locations.

The data base has information concerning three topological types: POINTs, ARCs, and REGIONs. A POINT is specified as the (two- or) three-space location of the entity; an ARC is a linear list of POINTs (and may have a specified width); a REGION is an area of space bounded by a closed ARC. ARCs may have subarcs and REGIONs may have subregions. These descriptions, in addition to being stored in the semantic data structure, are also stored in the geometric structure described below.

These topological descriptions are used where appropriate. For example, ground control points and buildings are stored as POINTs. Roads and coastlines are stored as ARCs. Cities and bodies of water are stored as REGIONs.

The current data base stores generic information about cities, roads, railroads, bridges, coastlines, rivers, lakes, bays, islands, hills, peninsulas, buildings, piers, channels, and airports. The complexity of descriptions spans a wide range. A lake, for example, is described simply as a body of water surrounded by land. The description of a bridge is at the more complex end of the spectrum. A bridge is described as a linear, segmented path over a collection of world entities. The OVERGOER must be an element of a restricted set of linear, cultural features, such as roads or railroads, and the UNDERGOERS must be from a larger set of cultural and cartographic features. There is typically one bridge segment for each of the UNDERGOERS. The description also contains data about the type of supporting structure and the mean height of each segment.

Instances of world entities in the current data base are drawn from the San Francisco Bay Area, and include various cities (including San Francisco, Oakland, and Berkeley) and bodies of water (e.g., San

Francisco Bav). and various lakes and reservoirs. The cities contain roads. railroads, and buildings. Coastlines. piers. and dock areas are specified where appropriate. Selected bridges (e.g.. the Golden Gate and the San Francisco-Oakland Bav Bridges) are described. In addition. the data base stores a large number of ground control points selected to cover the areas of interest.

3. Geometrical Data Base

Many entities in the semantic data base have a geometric structure. Roads. for example, are linear structures passing through certain specified areas. Cities have the properties of regions, including a defined area and the ability to contain other geometric entities within their boundaries.

Many questions that could be asked of an IU data base require the direct use of these geometric properties. For example. to answer the query: What roads are in Berkeley?, requires the system to determine the physical area covered bv the city of Berkelev. and then to determine which of the roads known to it pass through that area.

To facilitate the answer to questions such as this. the system organizes semantic entities with geometric structure into a special geometric data base. This geometric data base is made up of a set of tree structures called KDTREEs. patterned after the data structure described bv [4]. It is a bin structure where a bin is split into subbins whenever they get too full. Bins are split in such a way as to evenly distribute their contents.

The simplest KDTREE structure is one for storing points selected from a two-dimensional area. Initially. all of points are inserted into a single bin. stored at the root of the KDTREE. If the bin is too large (i.e.. has too many points). it is split into two subbins by comparing the first coordinate (say X) of each point with the median value for that coordinate over all points in the bin. If the X value of the point being tested is greater than the median X value (called the pivot), it is placed in the HIGH bin. otherwise it goes into the LOW bin. After all points have been tested in this manner, there will be two new bins of approximately equal size. These are stored as descendants of the node being split, and the original bin is deleted. The pivot is stored with the node as well.

This process is repeated on the new offspring bins, selecting a new coordinate each time. Thus, at the second level, the median Y value would be the pivot, and at the third level, X would be used again. The resulting structure is a well-balanced tree whose tip nodes contain the data-storage bins.

A new point is inserted into the tree by starting at the root node, comparing the appropriate coordinate value of the new point with the pivot, and recursively checking either the HIGH or the LOW offspring, until a tip node is reached. This is the bin which "covers" the point, and so the point is inserted here. If after insertion, the bin is too full, it is split as described above.

The system can determine which points in a KDTREE are contained in a specified rectangular area by first discovering which bins are overlapped by the rectangle. These are found by subdividing the rectangle, beginning at the root node, and splitting the rectangle at the pivot coordinate (if that coordinate falls within the rectangle). The subrectangles are split again at the next level, and so on to the tip nodes. When a tip node is reached, each point in the bin is tested to see if it falls within the sub-rectangle, and if so, will be returned when the process terminates.

As an example, ground control points (GCPs) are stored in a three-dimensional KDTREE. In order to find the closest GCP to a given location, P, the system first finds which bin P would fall into, and selects the nearest point, P1, in that bin. Since there may be a closer point in an adjacent bin, a square area centered about P, with a side length of twice the distance from P to P1 is created. Any points that fall within the square that were not previously checked are examined; if the nearest point to P from that set is closer than P1, that point is returned; otherwise, P1 is returned.

The KDTREE structure was extended to allow storage and retrieval of features that cover a finite area. such as ARCs and REGIONs. For these cases. a bounding rectangular volume is computed and any bin overlapped by that volume is further tested to see if the feature actually falls within it. If so. it is inserted into the bin.

When a bin is split. the pivot is computed as the median of the centers-of-gravity of the subvolumes for features in that bin. In this case. however. some bounding volumes will be split (such as the one which provided the pivot). and will appear in both the HIGH and the LOW offspring. It may even be the case that all of the volumes will have been split. and each new bin has the same number of entries as the original. If this happens. the system will attempt to select a new coordinate for the pivot. If all coordinates are checked and no improvement results. the bin cannot be split.

Using programs to compute the distance from a point to a linear or area feature. the system can determine the closest such feature to a selected point. This allows the the system to answer questions such as: *What road is nearest to the Transamerica Pyramid?* The question: *What roads are in Berkeley?* is answered by computing a bounding rectangle for the area associated with the city of Berkeley. and retrieving all known roads that are enclosed. using a similar process as described above for retrieving point features. These candidates are further checked to ensure they are actually within the boundary of the city.

KDTREEs are provided for most of the elements of interest in the data base. Specific trees include those for roads. GCPs. images. coastlines. and buildings.

REFERENCES

1. Barrow. H. G.. et al.. "Interactive Aids for Cartography and Photo Interpretation". in Artificial Intelligence--Research and Applications. Annual Progress Report to ARPA. Contract DAAG29-76-C-0012. Stanford Research Institute. Menlo Park. California (June 1976).

2. Barrow. H. G.. "Interactive Aids for Cartography and Photo Interpretation". Semiannual Technical Report to ARPA. Contract DAAG29-76-C-0012. Stanford Research Institute. Menlo Park. California (November 1976).

3. Barrow. H. G.. "Interactive Aids for Cartography and Photo Interpretation". Semiannual Technical Report to ARPA. Contract DAAG29-76-C-0057. Stanford Research Institute. Menlo Park. California (May 1977).

4. Bentley. J. L.. "Multidimensional Binary Search Trees Used for Assoiative Searching". CACM, Vol. 18. No. 9 (September 1975).

5. Hendrix. G. G.. "The LIFER Manual: A Guide to BuildingNatural Language Interfaces." Tech. Note 138. Stanford Reseach Institute. Menlo Park. California (February 1977).

6. Sacerdoti. E. D.. "Language Access to Distributed Data with Error Recovery", Proc. Fifth IJCAI. Vol. 1. pp. 196-202. Cambridge. Massachusetts (August 1977).

7. Garvey. T. D.. "Perceptual Strategies for Purposive Vision". Tech. Note No. 117. Stanford Research Institute. Menlo Park. California (1976).

8. Tenenbaum. J. M., Garvey. T. D.. Weyl. S. A., and Wolf, H. C., "ISIS: An Interactive Facility for Scene Analysis Research". Proc. Second International Joint Conference on Pattern Recognition. Lyngby-Copenhagen. Denmark. pp. 123-125. (August 1974).